НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему Розробка системи взаємодії індивідуальних сторінок викладачів та студентів. Підсистема: індивідуальні сторінки студентів

Виконав (-ла): студент (-ка) ____курсу, групи_____

Драгуцан Андрій Вадимович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник Доцент Недашківський Олексій Леонідович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Драгуцан Андрій Вадимович_____
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка системи взаємодії індивідуальних сторінок викладачів та студентів. Підсистема: індивідуальні сторінки студентів_____
_____

керівник роботи Доцент Недашківський Олексій Леонідович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 202__р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи сервіси, що надють можливість стоврювати на хмарні сервери, JVM та мова програмування JavaScript, Micronaut

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації інтеграції, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи_____
_____

5. Перелік ілюстративного матеріалу
1)Діаграма класів, 2)діаграма структури бази даних, 3) діаграма use case
_____

6. Дата видачі завдання ”___”_____20____ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Розробка структур окремих підсистем		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	Захист програмного продукту		
8.	Передзахист		
9.	Захист		

Студент _____ _____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

АНОТАЦІЯ

Мета роботи полягала у створені підсистеми індивідуальних сторінок студентів з можливістю комунікації викладача зі студентом. Всі обчислення проводяться на стороні сервера повертаючи клієнту отриманий результат. Користувацький додаток представляє собою графічний інтерфейс веб-сторінки. Ключові слова: хмарний сервіс, мікросервіси, Micronaut, REST, React, Message Broker, Object Storage. Записка містить 62 сторінки, 35 рисунки та 20 посилань.

ABSTRACT

The purpose of the work was to create subsystems of individual pages of students with the possibility of communication between teacher and student. All calculations are performed on the server side returning the result to the client. The user application is a graphical interface of a web page. Keywords: cloud service, microservices, Micronaut, REST, React, Message Broker, Object Storage. The note contains 62 pages, 35 figures and 20 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7

ВСТУП 8

1 ПОСТАНОВКА ЗАГАЛЬНОЇ ЗАДАЧІ 10

2 ПОСТАНОВКА ЗАДАЧІ РОБОТИ 12

3 АНАЛІЗ ЗАГАЛЬНОЇ ПРЕДМЕТНОЇ ГАЛУЗІ 13

3.1 Moodle 13

3.2 Система ЕК НТУУ «КПІ» 16

4 РОЗПОДІЛ ОБОВ'ЯЗКІВ МІЖ ВИКОНАВЦЯМИ 18

5 ЗАСОБИ РОЗРОБКИ ЗАГАЛЬНОЇ ПРОГРАМНОЇ СИСТЕМИ 19

5.1 Фреймворк Micronaut 20

5.1.1 Підтримка аспектно-орієнтованого програмування 21

5.1.2 HTTP клієнт 24

5.2 Бібліотека React.js 26

6 МЕТОДИ РОЗРОБКИ ПІДСИСТЕМИ 28

6.1 Мікросервіси 28

6.2 Комунікація між сервісами 32

7 АРХІТЕКТУРА СИСТЕМИ 35

7.1 Сервіс студентів 36

7.2 Сервіс розкладу 36

7.3 Сервер брокеру повідомлень 37

8 ПРОГРАМНА РЕАЛІЗАЦІЯ ПІДСИСТЕМИ 39

8.1 Структура проекту 39

8.2 Бізнес-модель 39

8.3 Структура бази даних 41

9 МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПІДСИСТЕМОЮ 42

ВИСНОВКИ 45

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46

ДОДАТОК А 48

ДОДАТОК Б 50

ДОДАТОК В 55

ЗАГАЛЬНІ ВІДОМОСТІ 58

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ 59
ОПИС ЛОГІЧНОЇ СТРУКТУРИ 60
ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ 61
ВИКЛИК І ЗАВАНТАЖЕННЯ 62
ВХІДНІ І ВИХІДНІ ДАНІ 63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JVM – Java Virtual Machine, Віртуальна машина Java.

DI – Dependency injection, Впровадження залежності.

REST – Representational State Transfer, підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.

AOP – aspect-oriented programming, Аспектно-орієнтоване програмування.

API – Application Programming Interface прикладний програмний інтерфейс.

HTTP — HyperText Transfer Protocol. Протокол передачі гіпер-текстових документів.

IDE – Integrated development environment. Інтегроване середовище розробки.

ВСТУП

На сьогоднішній день існує проблема комунікації та взаємодії між викладачами та студентами в умовах дистанційного навчання. І на разі, для вирішення цієї проблеми вони вимушені використовувати різне програмне забезпечення проведення навчального процесу: сервіси для спілкування, для поширення матеріалів курсу та для введення поточного контролю.

Сьогодні існує лише декілька систем, які частково вирішують дану проблему. Також більшість з них мають обмежений функціонал, який створений для покриття базових потреб, узагальнених серед всіх навчальних закладів, але при цьому унеможливлюється зміна чи додання нових функцій.

Тому метою даної роботи є вирішення поставленої проблеми шляхом запровадження розподіленої веб-система на базі мікросервісної архітектури з наступним її інтегруванням у сайт кафедри чи факультету.

Така система дистанційної комунікації між студентами та викладачами наразі є актуальною для її використання у період дистанційного навчання та подальшої підтримки учбового процесу. Система буде надавати як базові можливості: централізований доступ до сервісу через веб-браузер, зручний спосіб спілкування, можливість поширювати матеріали лекцій та переглядати студентами поточні оцінки - так і, завдяки мікросервісній архітектурі, легкому впровадженню нового фунціоналу, без обмежень у виборі мов програмування.

Данна підсистема веб-застосунку дозволятиме викладачу обмінюватися повідомленнями у чатах викладач-група, поширювати файли курсу на відведеній сторінці, переглядати розклад та виставляти поточний контроль.

Для реалізації серверної частини веб застосунку було вирішено використовувати фреймворк Micronaut. Його головними перевагами є асинхронність, швидке підняття серверу, мінімальне використовування оперативної пам’яті та велика кількість інструментів для створення швидких і функціональних мікросервісів. Даний фреймворк написаний на мові програмування Java, але також надає можливість використовувати його з такими мовами програмування як Kotlin та

Groovy.

Взаємодія між мікросервісами розроблена з використанням програмного брокера Kafka.

Для надання клієнтській частині сучасного вигляду та швидкої реалізації з можливістю створювати гнучкий інтерфейс було використано бібліотеку React яка базується на мові програмування JavaScript.

1 ПОСТАНОВКА ЗАГАЛЬНОЇ ЗАДАЧІ

Загальна мета даної роботи полягає у полегшенні комунікації між викладачами та студентами і підтримки учбового процесу шляхом впровадження системи взаємодії індивідуальних сторінок викладачів та студентів. Також додатковою метою є розробка другорядних підсистем для підтримки чатів, інтеграції з вже існуючою системою розкладу занять НТУУ «КПІ» ім. І. Сікорського, запровадження додаткового функціоналу та підтримки головних сервісів.

Для забезпечення повноти функціоналу та надійності були поставлені такі вимоги до загальної системи програмних засобів розроблених у роботі:

- можливість легкого горизонтального розширення кожної з підсистем;
- можливість легкого інтегрування нових підсистем;
- можливість створювати кластери з однотипних підсистем;
- виведення з ладу одного з екземплярів підсистеми в кластері, не повинно призводити до зупинки всього кластеру підсистеми;
- система повинна бути відмово стійкою.
- система повинна надавати можливість миттєвої передачі повідомлень між користувачами;
- можливість передавати повідомлення як між двома користувачами, так і в середині групи користувачів;
- можливість завантажувати, поширювати, завантажувати та безпечно зберігати файли;

Для досягнення мети та виконання вище перелічених вимог були поставленні та розв’язані наступні завдання:

- вибір фреймворку для реалізації мікросервісної архітектури;
- вибір архітектури мережевих протоколів для комунікації між клієнтом і сервісами та в середині хмари, між сервісами;
- вибір протоколу та його програмної реалізації для миттєвої передачі повідомлень;

- вибір бази даних для збереження об’єктів бізнес логіки;
- вибір об’єктного сховища для збереження файлів;
- розбиття додаткового функціоналу на логічні підсистеми та їх впровадження;

2 ПОСТАНОВКА ЗАДАЧІ РОБОТИ

За мету даної роботи було поставлено створення підсистеми індивідуальних сторінок викладачів, яка буде надавати можливості та функціонал для полегшення учбового процесу студенту, комунікації між студентами та викладачами, і розробці зручного та інтуїтивно зрозумілого графічного інтерфейсу.

Також були висунуті наступні вимоги до підсистема індивідуальних сторінок студентів:

- можливість завантажувати електронний матеріал у вигляді файлів;
- можливість для студента переглядати розклад занять;
- надання доступу до електронного матеріалу групам та студентам;
- можливість комунікації з одногрупниками та викладачем у групових чатах;
- можливість персональної комунікації з викладачем;
- зручний графічний інтерфейс.

Для досягнення мети та виконання вимог були поставленні та розв’язані наступні завдання:

- створення моделей бізнес логікі які описують предметну область;
- створення відображень та відношень між моделями даної підсистеми та моделями системи розкладу занять НТУУ «КПІ» ім. І. Сікорського;
- створення базового функціоналу для полегшення роботи студента
- розробка дизайну графічного інтерфейсу.

3 АНАЛІЗ ЗАГАЛЬНОЇ ПРЕДМЕТНОЇ ГАЛУЗІ

Системи підтримки учбового процесу та комунікації між її учасниками є достатньо вузькоспеціалізованим програмним забезпеченням, тому не являються розповсюдженими. Серед проаналізованих аналогів загальної системи даної роботи, були відібрані наступні системи: moodle та сайт підтримки навчального процесу НТУУ «КПІ» ім І. Сікорського ecampus.kpi.ua.

3.1 Moodle

Moodle(Modular Object-Oriented Dynamic Learning Environment) - це навчальна платформа, створена для того, щоб надати викладачам, адміністраторам та учням єдину надійну, безпечну та інтегровану систему для створення персоналізованих навчальних середовищ. Він надає можливість завантажити програмне забезпечення на власний веб-сервер. На основі Moodle створено десятки тисяч навчальних середовищ у всьому світі, Moodle довіряють великі та малі установи та організації, включаючи Shell, Лондонську школу економіки, Державний університет Нью-Йорка, Microsoft та Open University. Всесвітня кількість аудиторії Moodle - понад 90 мільйонів користувачів на рівні академічного та корпоративного рівня - це найпоширеніша навчальна платформа у світі. Завдяки 10-річному розвитку, керованому педагогікою соціальної конструктивістики, Moodle пропонує потужний набір інструментів, орієнтованих на учнів, та викладачів, що сприяють навчанню та викладанню. Простий інтерфейс, функції перетягування та добре задокументовані ресурси, а також постійні удосконалення роблять Moodle легким у вивченні та використанні.

Moodle надається безкоштовно як програмне забезпечення з відкритим кодом, відповідно до Загальної публічної ліцензії GNU. Будь-хто може адаптувати, розширювати або змінювати Moodle як для комерційних, так і для некомерційних проектів без будь-яких ліцензійних платежів та отримувати вигоду від економічної ефективності, гнучкості та інших переваг використання Moodle. Підхід проекту з відкритим кодом Moodle означає, що Moodle постійно переглядається та вдосконалюється, щоб відповідати поточним та розвиваючим потребам своїх

користувачів.

Рисунок 3.1 – Інтерфейс системи moodle

Багатомовні можливості Moodle забезпечують відсутність лінгвістичних обмежень для навчання в інтернеті. Спільнота Moodle розпочала переклад Moodle більш ніж на 120 мов, щоб користувачі могли легко локалізувати свій сайт Moodle, а також безліч ресурсів, підтримки та обговорень спільноти на різних мовах. Moodle надає найбільш гнучкий набір інструментів для підтримки як змішаного навчання, так і повністю онлайн-курсів. Налаштуйте Moodle, включивши або відключивши основні функції, та легко інтегруйте все необхідне для курсу, використовуючи повний спектр вбудованих функцій, включаючи зовнішні інструменти спільної роботи, такі як форуми, вікі, чати та блоги. Оскільки він є відкритим кодом, Moodle може бути налаштований будь-яким способом та підлаштовуватися під індивідуальні потреби. Його модульне налаштування та сумісна конструкція дозволяє розробникам створювати плагіни та інтегрувати зовнішні програми для досягнення конкретних функціональних можливостей. Розширюйте те, що робить Moodle, використовуючи вільно доступні плагіни та додатки. Від декількох студентів до мільйонів користувачів Moodle може бути розширений, щоб підтримувати потреби як малих класів, так і великих організацій. Завдяки своїй гнучкості та масштабованості Moodle

був пристосований для використання в контексті освіти, бізнесу, некомерційного, урядового та громадського середовища. Прихильний до захисту безпеки даних та конфіденційності користувачів, контроль безпеки постійно оновлюється та впроваджується в процеси розробки та програмного забезпечення Moodle для захисту від несанкціонованого доступу, втрати даних та неправильного використання. Moodle може бути легко розгорнутий на приватному захищеному сервісі-хмарі або сервері для повного контролю.

Moodle базується на веб-серверній архітектурі, тому до нього можна отримати доступ з будь-якої точки світу. Завдяки інтерфейсу для мобільних пристроїв за замовчуванням та сумісності між браузерами вміст на платформі Moodle легко доступний та сумісний у різних веб-браузерах та пристроях. Отримайте доступ до великої документації щодо Moodle та форумів користувачів на різних мовах, безкоштовного контенту та курсів, якими користуються користувачі Moodle у всьому світі, а також сотень плагінів, внесених великою світовою спільнотою.

Проект Moodle підтримується активною міжнародною спільнотою, командою відданих розробників та штатних партнерів Moodle. Керуючись відкритою співпрацею та великою підтримкою громади, проект продовжує досягати швидких виправлень помилок та вдосконалень, з новими основними випусками кожні півроку.

Як переваги даної системи можна перелічити наступні пункти:

- Сучасний, простий у користуванні інтерфейс;
- Персоналізована інформаційна панель;
- Інструменти та заходи для спільної роботи;
- Календар;
- Зручне управління файлами;
- Простий та інтуїтивно зрозумілий текстовий редактор;
- Сповіщення;
- Відстеження прогресу.

Серед недоліків можна виділити наступне:

- Немає зручної системи для миттєвої комунікації між студентами та студентами і викладачами;
- Цілодобова підтримку користувачів, яка через велику кількість запитів не завжди має можливість швидко відповісти.

3.2 Система ЕК НТУУ «КПІ»

Електронний кампус НТУУ «КПІ» ecampus.kpi.ua є складовою Корпоративного порталу НТУУ «КПІ», що є підсистемою Єдиного Інформаційного Середовища НТУУ «КПІ». Система була розроблена для інформаційної підтримки навчального процесу університету з метою підвищення якості навчання студентів, забезпечення навчального процесу сучасними інформаційними технологіями. Серед функціоналу система надає наступні рішення. Віртуальні кабінети за профілями користувачів: студент, що розширює функціонал, якщо студент є старостою чи профоргом; викладач-науковець, розширюється якщо профіль є куратором групи; методист кафедри; завідувач кафедрою. Роботу з ЕК НТУУ «КПІ» підтримують всі сучасні браузери, такі як Internet Explorer, Mozilla Firefox, Opera і Google Chrome. Головна сторінка ПВК системи ЕК НТУУ «КПІ» містить кнопки перходу до наступних розділів: мій профіль, щоденник, контакти, довідка, форум, дошка оголошення, повідомлення та розкладу (рис. 3.2).

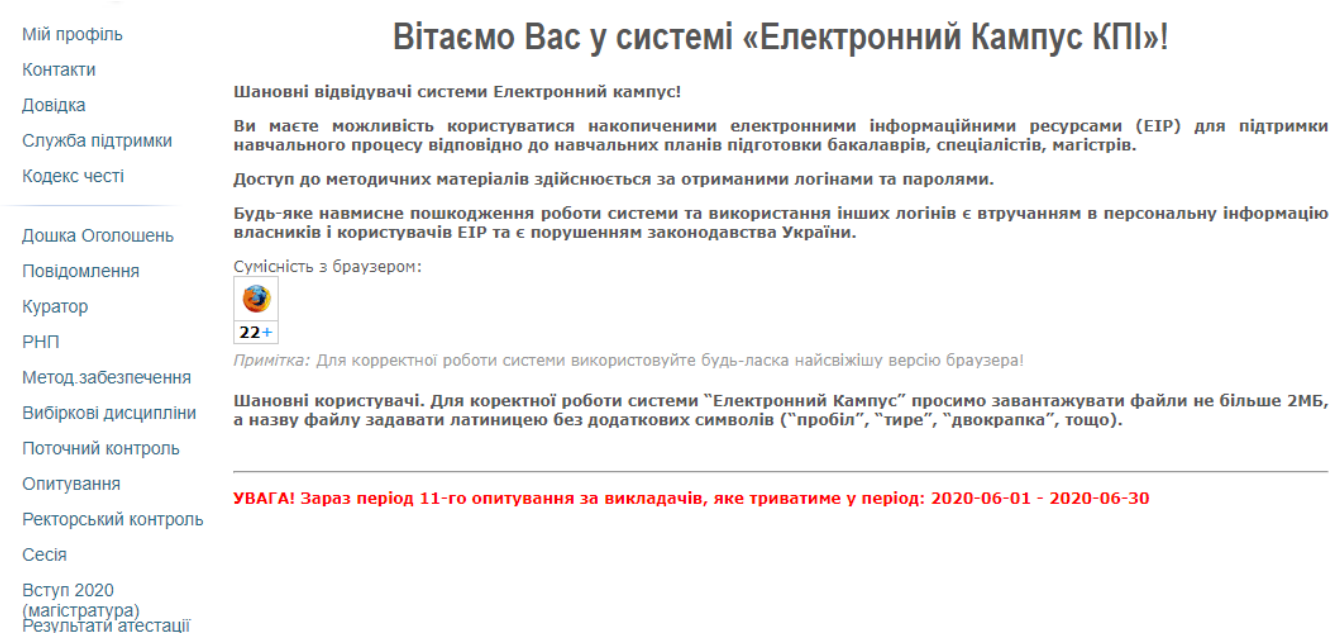

Рисунок 3.2 – Головна сторінка ПВК системи ЕК НТУУ «КПІ»

Серед переваг системи можна виділити наступне:

- Простий для розуміння інтерфей;
- Доступ до усієї потрібної інформації такої як поточного контролю, контактів, довідки, форуму, дошки оголошень та розкладу;
- Анонімна система оцінювання викладачів.

Серед недоліків ЕК НТУУ «КПІ» є такі як:

- Не повний функціонал усіх вкладок;
- Недороблена основна версія кампусу (поточна робота з електронним кампусом ведеться у тимчасовій версії);
- Технічні помилки в інтерфейсі.

Кожна з систем має свої недоліки, через що, повнота функціоналу не задовольняє сучасним потребам до систем інформаційної підтримки навчального процесу університету.

Також, спільним недоліком є відсутність або погано реалізована підсистема комунікації між студентами в групах та студентами і викладачами. З огляду на це, для вдосконалення функціоналу об’єктів даної предметної галузі, було вирішено впровадити сучасну реалізацію миттєвого обміну між користувачами та збереження корисних можливостей існуючих систем.

4 РОЗПОДІЛ ОБОВ'ЯЗКІВ МІЖ ВИКОНАВЦЯМИ

Так-як реалізуєма система складається з двох головних підсистем та декількох допоміжних, було поставлено завдання розподілу обов’язків між її виконавцями. Головними задачами до розподілу обов’язків є мінімізування загального часу реалізації всієї системи та розробка кожним виконавцем сервісів, в реалізувані яких він має більше досвіду.

Виходячи з зазначеного вище було прийнято рішення по розподілу обов’язків наступним чином. Перший виконавець відповідальний за розробку таких сервісів та підсистем:

- Підсистема індивідуальних сторінок викладачів;
- Сервіс автентифікації;
- Сервер об’єктного сховища.

Другий виконавець має реалізувати наступні елементи системи:

- Підсистема індивідуальних сторінок студентів;
- Сервіс розкладу;
- Сервер брокеру повідомлень.

5 ЗАСОБИ РОЗРОБКИ ЗАГАЛЬНОЇ ПРОГРАМНОЇ СИСТЕМИ

Для реалізації серверної частини загальної програмної системи було обрано мову програмування Java[1]. Серед всіх фреймворків даної мови програмування було проаналізовано та обрано два, які розраховані на створення серверних додатків: Spring[2] та Micronaut[3]. Spring являється одним з основних фреймворків для створення серверних додатків, так-як він має широкий спектр для вирішення базових задач та існує використовується з 2004 року. Micronaut є новим фреймворком, так як він існує з 2019 року, але попри це, як і Spring, має достатній функціонал та інтеграцію з іншими серверними додатками. Порівнюючи ці фреймворки, було вирішено використовувати Micronaut, так-як як він має наступуні переваги, які дозволяють легше прототипувати мікросервісну архітектуру:

- інструменти для автоматичного визначення середовища та подальшого налаштування додатку;
- інструменти для легкої конфігурації додатка, в залежності від середовища розгортання;
- базова підтримка як асинхронної так і синхронної обробки запитів;
- інтеграція з бібліотеками та додатками брокерів повідомлень;

Для тестування було обрано бібліотеку Spock[4]. Ця бібліотека дозволяє використовувати підхід behaviour-driven development(з англ. - керована поведінкою розробка). Цей підхід дозволяє тестувати не конкретний об’єкт чи метод, а деяку поведінку чи специфікацію.

Клієнтську частину системи було вирішено реалізовувати на мові програмування JavaScript[5] з використанням бібліотеки React[6], яка є бібліотекою програмних засобів з відкритим вихідним кодом, розробленою компанією Facebook. Дана бібліотека дозволяє створювати графічний інтерфейс, що складається з програмних одиниць - «компонентів». Вони можуть бути впроваджені розробником або іншими бібліотеками. Дана бібліотека дозволяє використовувати один компонент у багатьох інших компонентах або сторінках. Такий підхід дозволяє пришвидшити прототипування дизайну клієнтської частини і відділити функціональну логіку

обробки графічного інтерфейсу, від логіки комунікації з серверною частиною.

Так-як серверна частина побудована за принципом мікросервісної архітектури, важливим елементом стає брокер повідомлень, який виконує функцію забезпечення передачі повідомлень між мікросервісами. В ситуації коли відправлене повідомлення одного сервісу може бути не прийнятим іншим, перевантаженим, сервісом, воно втрачається, що призводить до втрати важливої інформації. Тому для комунікацію між сервісами використовують брокер повідомлень, який гарантує передачу повідомлень. В даній системі було вирішено використовувати брокер повідомлень Apache Kafka[7] розроблений компанією LinkedIn. Його головними перевагами є легке горизонтальне розширення та тимчасового зберігання даних, що були передані.

5.1 Фреймворк Micronaut

Micronaut – фреймворк на базі JVM для побудови модульних, з легким тестуванням мікросервісних та безсерверних додатків. При його проектуванні, розробники мали на меті забезпечити всі інструменти, необхідні для створення повнофункціональних мікросервісних додатків, включаючи:

- Продумані значення за замовчуванням та автоматична конфігурація
- Можливість конфігурації та поширення конфігурацій
- Знаходження сервісів
- Маршрутизація HTTP
- HTTP-клієнт з балансуванням навантаження на стороні клієнта

У той же час Micronaut уникнає недоліків інших фреймворків, забезпечуючи:

- Швидкий час запуску
- Зниження відбитку використовування пам'яті
- Мінімальне використання рефлексії
- Мінімальне використання проксі
- Легке блочне тестування

Також такі фреймворки, як Spring та Grails[8], не розроблялися для використання

у таких сценаріях, як безсерверних функції, додатки для Android або мікросервіси з низьким використанням пам’яті. На відміну від них, Micronaut розроблений таким чином, щоб він підходить для всіх цих сценаріїв.

Ця мета досягається завдяки використанню процесорів анотацій Java, які можна використовувати на будь-якій мові JVM, яка їх підтримує та використанню HTTP-серверу та клієнту, побудованих на Netty[9]. Для того, щоб надати подібну модель програмування як у Spring чи Grails, ці процесори анотацій попередньо компілюють необхідні метадані для виконання DI, визначення AOP проксі та налаштування додатка для роботи в середовищі мікросервісів.

5.1.1 Підтримка аспектно-орієнтованого програмування

Аспектно-орієнтоване програмування (англ. aspect-oriented programming – AOP) являє собою парадигму програмування, яка спрямована на підвищення модульності за рахунок можливості поділу наскрізних проблем. Вона робить це шляхом додавання додаткової поведінки до існуючого коду порад (advices) без модифікації самого коду. Натомість, щоб окремо вказати, який код модифікується, використовується специфікація точки зрізу (pointcut). Наприклад, "реєструвати всі виклики функцій, якщо ім'я функції починається з 'set ' ". Це дозволяє додавати в програму поведінку, що не є головною для бізнес-логіки (наприклад, логування), не забруднюючи код, який є ядром функціональності. AOP формує основу для аспектно-орієнтованої розробки програмного забезпечення. AOP включає в себе методи і інструменти програмування, які підтримують модулярізацію проблем на рівні вихідного коду, в той час як "аспектно-орієнтована розробка програмного забезпечення" відноситься до цілої інженерної дисципліни.

Аспект-орієнтоване програмування має на меті розбиття програмної логіки на окремі частини (так звані проблеми, зв'язкові області функціональності). Майже всі парадигми програмування підтримують певний рівень групування і інкапсуляції проблем в окремі самостійні об’єкти, надаючи абстракції (наприклад, функції, процедури, модулі, класи, методи), які можуть бути використані для реалізації, абстрагування і компонування цих проблем. Деякі з них "перетинають" кілька абстракцій в програмі і кидають виклик цим формам реалізації. Ці проблеми

називаються "наскрізними" або "горизонтальними".

Протоколювання (logging) є прикладом наскрізного підходу, оскільки стратегія протоколювання обов'язково зачіпає кожну записану в журнал частина системи. Таким чином, протоколювання зачіпає всі класи і методи протоколювання.

Аспект-орієнтоване програмування історично мало багато втілень і деякі дуже складні реалізації. Зазвичай AOP можна розглядати як спосіб вирішення наскрізних проблем (протоколювання, транзакції, трасування і т.д.) окремо від коду програми у вигляді аспектів, що визначають поради. Як правило, існує дві форми порад:

- Around Advice – декорує метод або клас.
- Introduction Advice – вводить в метод нову поведінку.

На рисунку 5.1 показано реалізацію Introduction Advice за допомогою бібліотек фреймворку Micronaut.

```
import io.micronaut.aop.*;
import javax.inject.Singleton;

@Singleton
public class StubIntroduction implements MethodInterceptor<Object,Object> { (1)

    @Override
    public Object intercept(MethodInvocationContext<Object, Object> context) {
        return context.getValue( (2)
                Stub.class,
                context.getReturnType().getType()
        ).orElse(null); (3)
    }
}
```

Рисунок 5.1 – Приклад реалізації Introduction Advice. Цифрами позначено: (1) Клас повинен реалізовувати інтерфейс MethodInterceptor. (2) Отримання значення, яке буде задане як параметр даної анотації. (3)В іншому випадку повертати null.

В сучасних Java-додатках декларування порад, як правило, приймає форму анотації. Найвідомішим порадою в світі Java, ймовірно, є @Transactional, яка використовується для демаркації меж транзакцій в додатках Spring і Grails. Недоліком традиційних підходів до AOP є сильна залежність від створення і відображення проксі

під час виконання програми, що уповільнює продуктивність програми, ускладнює налагодження і збільшує споживання пам'яті. Micronaut намагається вирішити ці проблеми, надаючи простий AOP API часу компіляції, який не використовує відображеннь.

Micronaut AOP API також надає низку корисних реалізованих порад. Одна з таких, анотація Schedule (планування), що може бути додана в будь-який метод класу. Для коректної роботи, розробник повинен вказати одне з полей анотації: fixedRate, fixedDelay або cron. Приклад застосування наведений на рисунку 5.2.

```
@Scheduled(fixedRate = "5m")
void everyFiveMinutes() {
    System.out.println("Executing everyFiveMinutes()");
}
```

Рисунок 5.2 – Застосування анотації Schedule у коді програми

Також Micronaut надає можливості кешування через AOP API. Стандартним програмним забезпеченням для кешування є бібліотека Caffeine, але фреймворк підтримує і інші популярні програмні додатки. Для використання функціоналу, Micronaut має наступні анотації кешу:

- @Cacheable - вказує на те, що метод кешується всередині заданого імені кешу.
- @CachePut - Вказує, що значення що повертається з виклику методу повинно бути збережено в кеші. На відміну від @Cacheable, виклик метод ніколи не пропускається.
- @CacheInvalidate - Вказує, що виклик методу повинен привести до інвалідації одного або декількох кешей.

При використанні однієї з анотацій активується CacheInterceptor, який в разі @Cacheable буде кешувати дані, які повертає метод.

Якщо тип методу є неблокуючим (або CompletableFuture, або реалізація Publisher), то отриманий результат буде кешуватися. Крім того, якщо базова реалізація Cache підтримує неблокуючі кеш-операції, то значення в кеші будуть зчитуватися без блокування, в результаті чого з'являється можливість реалізувати

повністю не блокуючі кеш-операції.

```
@Singleton (1)
@CacheConfig("headlines") (2)
public class NewsService {

    Map<Month, List<String>> headlines = new HashMap<Month, List<String>>() {{
        put(Month.NOVEMBER, Arrays.asList("Micronaut Graduates to Trial Level in Thoughtworks technology radar Vol.1",
                "Micronaut AOP: Awesome flexibility without the complexity"));
        put(Month.OCTOBER, Collections.singletonList("Micronaut AOP: Awesome flexibility without the complexity"));
    }};

    @Cacheable (3)
    public List<String> headlines(Month month) {
        try {
            TimeUnit.SECONDS.sleep(3); (4)
            return headlines.get(month);
        } catch (InterruptedException e) {
            return null;
        }
    }

    @CachePut(parameters = {"month"}) (5)
    public List<String> addHeadline(Month month, String headline) {
        if (headlines.containsKey(month)) {
            List<String> l = new ArrayList<>(headlines.get(month));
            l.add(headline);
            headlines.put(month, l);
        } else {
            headlines.put(month, Arrays.asList(headline));
        }
        return headlines.get(month);
    }
```

Рисунок 5.3 - Приклад реалізації кешу

5.1.2 HTTP клієнт

Так-як даній підсистемі потрібно буде звертатись до віддалених серверів інших зовнішніх сервісів, фреймворк повинен мати надійну імплементацію HTTP клієнту. Критичним компонентом будь-якої архітектури мікросервісів є клієнтська комунікація між сервісами. З огляду на це, Micronaut має вбудований HTTP клієнт, який має як низькорівневий API, так і API більш високого рівня, керований AOP.

Інтерфейс HttpClient є основою низькорівневого API. Цей інтерфейс декларує методи, що полегшують виконання HTTP-запитів і отримання відповідей. Більшість методів в інтерфейсі HttpClient повертає екземпляри Reactive Streams Publisher, який не завжди є найбільш корисним інтерфейсом для роботи, тому в нього включений підінтерфейсів під назвою RxHttpClient, який забезпечує варіацію інтерфейсу HttpClient, що повертає типи RxJava Flowable.

Є кілька способів отримати об’єкт HttpClient. Найбільш поширеним способом є використання анотації Client. Анотація Client також може настроюватися областю, яка буде керувати створенням примірників HttpClient і забезпечувати їх вивантаження

при завершенні роботи програми. Значення, яке ви передаєте в анотацію клієнта, може бути одним з наступних:

- Абсолютний URI. Приклад https://api.twitter.com/1.1.
- Відносний URI, в цьому випадку цільовим сервером буде поточний сервер.
- Ідентифікатор сервісу.

Іншим способом створення HttpClient є створення статичного методу RxHttpClient, проте такий підхід не рекомендується, так як вам доведеться вручну відключити клієнт та не буде можливості ін'єкції залежності для створеного клієнта. Як правило, при роботі з HttpClient інтерес представляють два методи. Перший метод називається retrieve, який виконує HTTP-запит і повертає тіло того типу, який ви запитуєте (за замовчуванням String) у вигляді RxJava Flowable. Метод retrieve приймає HttpRequest об'єкт або String URI до кінцевої точки, на яку ви хочете відіслати запит. На рисунку 5.4 наведений приклад що показує, як використовувати retrieve для виконання HTTP GET запиту і отримання тіла відповіді у вигляді String.

```
String uri = UriBuilder.of("/hello/{name}")
                       .expand(Collections.singletonMap("name", "John"))
                       .toString();
assertEquals("/hello/John", uri);

String result = client.toBlocking().retrieve(uri);

assertEquals(
        "Hello John",
        result
);
```

Рисунок 5.4 – Надсилання запиту за допомогою HttpClient

Також, є можливість використовувати високорівневе API, для створення HttpClient. Для цього існує анотація Client, яка може бути оголошена на будь-якому інтерфейсі або абстрактному класі, і за допомогою Introduction Advice абстрактні методи будуть реалізовані для вас на етапі компіляції, що значно спростить створення HTTP клієнтів. Приклад застосування наведений на рисунку 5.5.

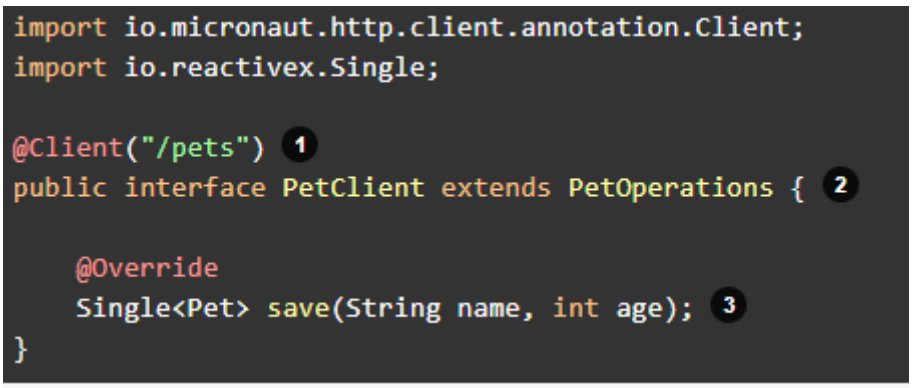

Рисунок 5.5 – Застосування анотації Client у коді програми. Цифрами позначено: (1) Анотація Client використовується зі значенням щодо поточного сервера. В цьому випадку /pets. (2) Інтерфейс розширено за рахунок PetOperations. (3)Метод save перевизначений.

5.2 Бібліотека React.js

Для розробки клієнтської частини було вирішено створити веб-застосунок по принципу односторінкового застосунку SPA (single-page application). Такі веб-застосунки чи веб-сайти, які вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою. В односторінковому застосунку весь необхідний код - HTML, JavaScript, та CSS - завантажується разом зі сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею. Взаємодія з односторінковим застосунком часто включає в себе динамічний зв'язок з веб-сервером.

Для реалізації даної парадигми було вирішено використовувти бібліотек React.js. React.js це відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні

модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно (diff) зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити.

Компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP.

6 МЕТОДИ РОЗРОБКИ ПІДСИСТЕМИ

Так-як засоби розробки програмного забезпечення що описані у попередньому розділі дозволяють досягти мети даної роботи декількома шляхами, що полягає у розробці підсистеми полегшення навчального процесу та комунікації між студентом і викладачем, та були висунуті специфічні вимоги, було вирішено використовувати наступні методи розробки програмного забезпечення.

6.1 Мікросервіси

Для розробки якісної підсистеми, перед початком реалізації програмного продукту, постає задача у виборі загальної архітектури системи. Аналіз задач та вимог дозволяє визначитись з функціями системи, які зазвичай можуть бути не сумісними між собою при стандартних реалізаціях. Виходячи з того, що дана система повинна використовувати такі підходи як REST API, підтримувати протокол WebSocket та кожна зі підсистем повинна мати можливість комунікувати одна з одною – реалізувати стандартну монолітну архітектура не можливо. Тому було вирішено використовувати мікросервісну архітектуру.

Мікросервіси – також відомі як архітектура мікро-послуг - це архітектурний стиль, який структурує додаток як колекцію послуг, які є зручні в підтримці і тестуванні, незалежно пов'язані, мають можливість незалежного розгортання та можуть розроблятися невеликою командою. Архітектура мікросервіса забезпечує швидку, часту і надійну доставку великих, складних додатків. Вона також дозволяє організації розвивати свій технологічний стек. Головним є піхід до розробки однієї програми в вигляді набору невеликих служб, кожна з яких працює в своєму власному процесі і взаємодіє з легковажними механізмами, часто з HTTP ресурсним API. Ці сервіси побудовані навколо можливостей бізнесу і можуть бути незалежно розгорнуті за допомогою повністю автоматизованого обладнання для розгортання. Існує абсолютний мінімум централізованого управління цими сервісами, які можуть бути написані на різних мовах програмування і використовувати різні типи технологій зберігання даних.

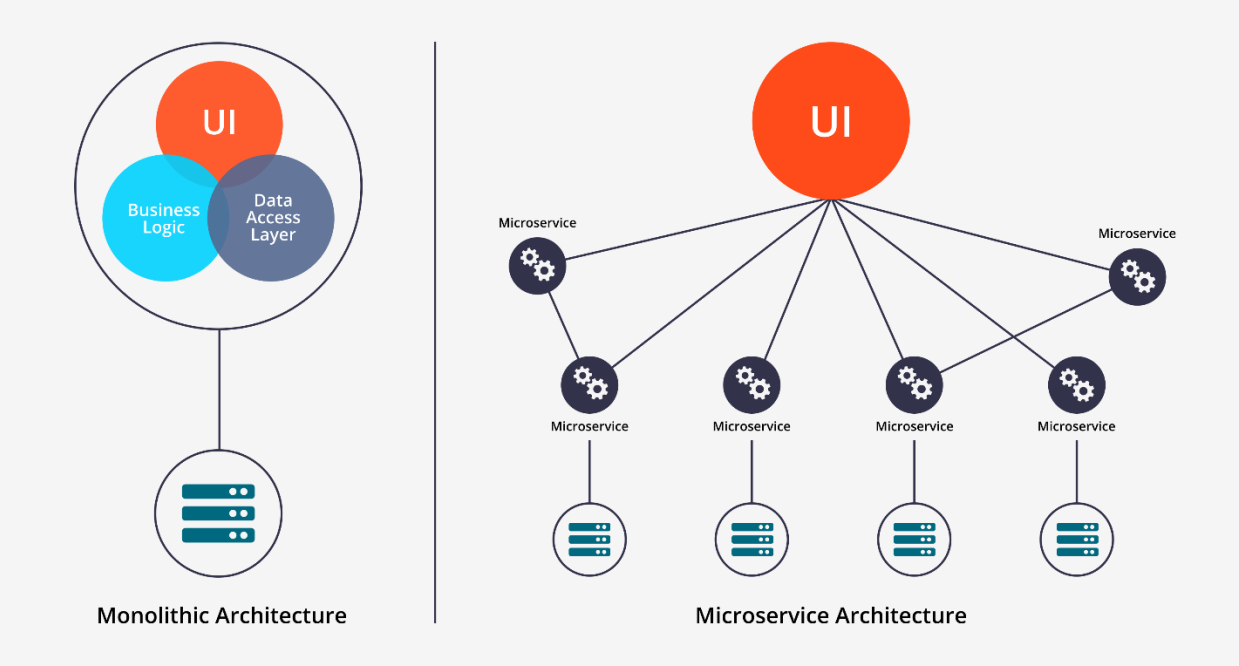

Рисунок 6.1 – Монолітна та мікросервісна архітектури

Щоб пояснити стиль мікросервісів, наочним буде порівняння його з монолітним стилем: монолітний додаток, побудоване як єдине ціле. Корпоративні додатки часто складаються з трьох основних частин: користувальницького інтерфейсу на стороні клієнта (що складається з HTML-сторінок і javascript, виконуваного в браузері на машині користувача), бази даних (що складається з безлічі таблиць, що вставляються в загальну, і зазвичай реляционную, систему управління базами даних) і серверного додатка. Серверний додаток буде обробляти HTTP-запити, виконувати логіку домену, витягувати і оновлювати дані з бази даних, а також вибирати і заповнювати HTML-вистави для відправки в браузер. Такий серверний додаток є монолітом - єдиним логічним виконуваним файлом. Будь-які зміни в системі призводять до створення і розгортання нової версії серверного додатка.

Такий монолітний сервер став стандартним підходом до побудови систем. Вся логіка обробки запиту виконується в єдиному процесі, що дозволяє використовувати основні можливості мови програмування для поділу додатки на класи, функції і простору імен. З певною обережністю є можливість запустити і протестувати додаток на ноутбуці розробника, а також використовувати конвеєр розгортання, щоб переконатися, що зміни належним чином протестовані і впроваджені. Горизонтальне

масштабування моноліту виконується через запуск нового екземпляра з розподільником навантаження. Монолітні додатки можуть успішно використовуватись для рішення різних проблем, але останнім часом з впровадженням хмарних технологій, до яких вони не пристосовані, відходять у сторону. Цикли змін прив'язані один до одного - зміна, внесена в невелику частину програми, вимагає перебудови і розгортання всього моноліту. З плином часу часто буває складно зберегти хорошу модульну структуру, що ускладнює впровадження змін, які повинні торкатися лише один модуль всередині цього модуля. Масштабування вимагає масштабування всієї програми, а не його частин, що вимагають великих ресурсів.

Ці недоліки привели до архітектурного стилю мікро-сервісу: будівництво додатків в якості комплексу сервісів. Крім того, що сервіси незалежно розгортаються і масштабуються, кожен сервіс також забезпечує жорстку межу між модулями, дозволяючи писати різні сервіси на різних мовах програмування. Вони також можуть управлятися різними командами. Мікросервісна архітектура має свої характеристики та ознаки, але провадження всіх не є обов’язковим.

Головною ознакою мікросервісів є компонентність. Компонент - це одиниця програмного забезпечення, яка незалежно замінюється і оновлюється. Мікросервісні архітектури використовують бібліотеки, але їх основним способом компонування власного програмного забезпечення є розбиття на сервіси. Ми визначаємо бібліотеки як компоненти, які пов'язуються в програму і викликаються за допомогою функцій внутрішньої пам'яті, в той час як служби є внепроцесснимі компонентами, які взаємодіють з між собою такими способами як запити веб-служб або віддалений виклик процедур. Одна з основних причин використання служб як компонентів (а не бібліотек) полягає в тому, що служби можуть розгортатися незалежно один від одного. Якщо у вас є додаток, що складається з декількох бібліотек в одному процесі, то зміна будь-якого одного компонента призводить до необхідності перезапуску всієї програми. Але якщо цей додаток розбити на кілька сервісів, можна очікувати, що багато змін в одному сервісі вплинуть тільки на цей сервіс. Але працює не у всіх випадках, деякі зміни змінять сервісні інтерфейси, що призведе до деякої переробки функціоналу в інших, але метою хорошою архітектури мікро-сервісу є мінімізація

цих змін за допомогою узгоджених меж сервісів і механізмів еволюції в сервісних контрактах. Ще одним наслідком використання сервісів в якості компонентів є більш чіткий компонентний інтерфейс. Більшість мов не мають хорошого механізму для визначення явного публічного інтерфейсу. Також в більшості випадків дозволяють клієнтам порушити інкапсуляцію компонента, що призводить до занадто тісному зв'язку між компонентами. Сервіси дозволяють уникнути цього, використовуючи явні механізми віддаленого виклику. Використання подібних сервісів має свої недоліки. Дистанційні виклики займають більше часу та використовують більші ресурси, ніж внутрішньопроцессні. Якщо вам потрібно змінити розподіл обов'язків між компонентами, то такі зміни поведінки складніше зробити, коли перетинаються межі процесу. З першого погляду можна помітити, що сервіси зіставляються з виконавчими процесами, але це перший погляд. Сервіс може складатися з декількох процесів, які завжди будуть розроблятися і розвиватися разом, наприклад, процес додатку і бази даних, яка використовується тільки цим сервісом.

Іншою характеристикою мікросервісної архітектури є децентралізація управління даними. Децентралізація управління даними представлена в ряді різних програмних засобів. На самому абстрактному рівні це означає, що концептуальна модель предметної області буде відрізнятися між системами. Це загальна проблема, коли при інтеграції в рамках великого підприємства модель клієнта зі сторони замовлень буде відрізнятися від моделі з боку підтримку. Деякі параметри першої моделі можуть не з’явитись в другій, так і навпаки. Тому постає задача розділення даних, так-як при наявності спільних атрибутів, моделі можуть відрізнятись семантично. Ця проблема поширена між додатками, але також може виникати всередині додатків, зокрема, коли додаток розділений на окремі компоненти. Корисним способом вирішення цієї проблеми є концепція Domain-Driven Design (Дизайн, орієнтований на домен) - Bounded Context (Зв'язаний контекст). DDD розділяє складний домен на кілька обмежених контекстів і визначає відносини між ними. Цей процес корисний як для монолітної, так і для мікро-сервісній архітектури, проте існує природна кореляція між межами сервісу та контексту, яка допомагає пояснити і посилити поділ.

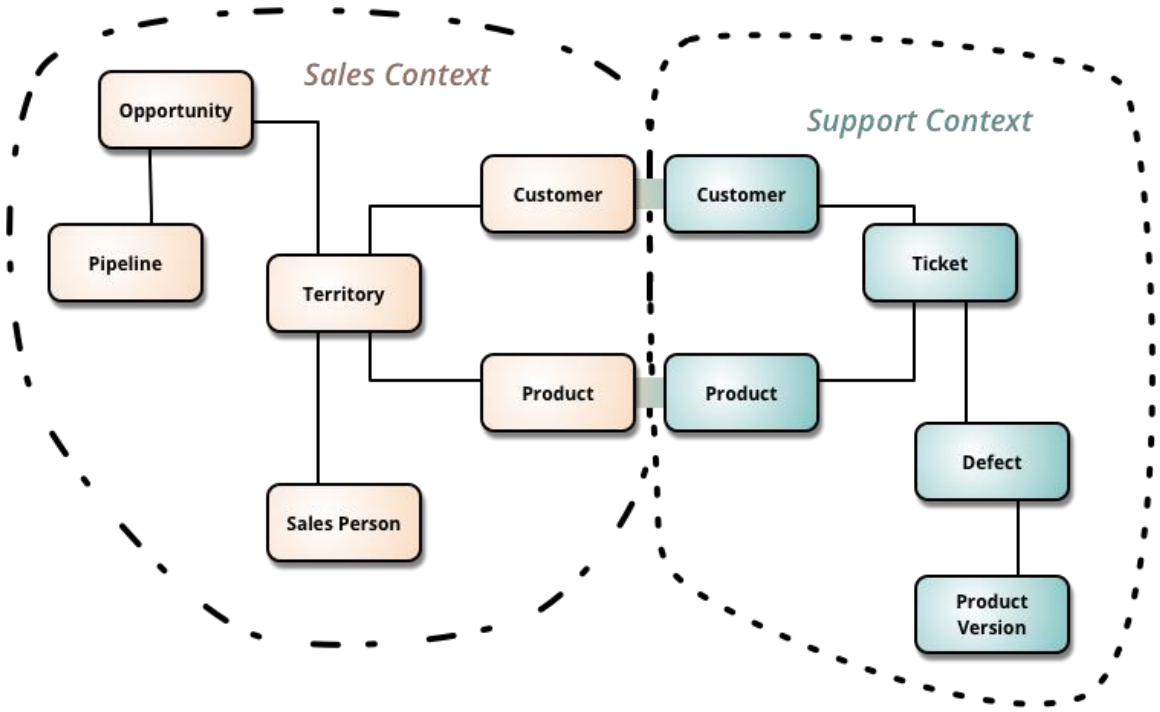

Рисунок 6.2 – Принцип Domain-Driven Design

Поряд з децентралізацією рішень по концептуальним моделям, мікросервіси також децентралізують зберігання даних. У той час як монолітні додатки переважно використовують єдину логічну базу даних для даних, в масштабах підприємства часто використовують єдину базу даних для цілого ряду додатків. В мікросервісній архітектурі прийнято, щоб кожна служба керувала власною базою даних, або різними екземплярами однієї й тієї ж баз даних реалізованих різними провайдерами – такий підхід, називається Polyglot Persistence. Polyglot Persistence можна використовувати в монолітному режимі, але вона частіше проявляється в мікросервісах. Децентралізація відповідальності за дані в мікрослужбах має наслідки для управління оновленнями. Загальний підхід до роботи з оновленнями полягає в використанні операцій для забезпечення узгодженості при оновленні декількох ресурсів. Цей підхід часто використовується в монолітах.

6.2 Комунікація між сервісами

Так-як додаток на основі мікросервісів - це розподілена система, що працює на кількох процесах або сервісах, зазвичай навіть на декількох серверах або хостах, кожен екземпляр сервісу повинен взаємодіяти, використовуючи протокол взаємодії між процесами, такий як HTTP, AMQP, або двійковий протокол, такий як TCP, в

залежності від характеру кожної служби. Клієнт і сервіси можуть спілкуватися за допомогою безлічі різних видів зв'язку, кожен з яких націлений на різні сценарії і задачі. Спочатку ці типи зв'язку можуть бути класифіковані по двох осях. Перша вісь визначає, чи є протокол синхронним або асинхронним:

- Синхронний протокол. HTTP є синхронним протоколом. Клієнт посилає запит і чекає відповіді від служби. Це не залежить від виконання клієнтського коду, який може бути синхронним (потік заблокований) або асинхронним (потік не заблокований, і відповідь в кінці кінців досягне зворотного виклику). Важливим моментом тут є те, що протокол (HTTP / HTTPS) є синхронним і код клієнта може тільки продовжити своє завдання, коли він отримує відповідь HTTP-сервера.
- Асинхронний протокол. Інші протоколи, такі як AMQP (протокол, підтримуваний багатьма операційними системами і хмарними середовищами), використовують асинхронні повідомлення. Клієнтський код або відправник повідомлення зазвичай не чекає відповіді. Він просто відправляє повідомлення, як при відправці повідомлення в чергу RabbitMQ або в будь-який інший поштовий брокер.

Друга вісь визначає, чи має комунікаційний зв’язок один приймач або кілька приймачів:

- Один приймач. Кожен запит повинен бути оброблений рівно одним приймачем або службою. Прикладом такого зв'язку є шаблон "Команда".
- Кілька приймачів. Кожен запит може бути оброблений від нуля до декількох приймачів. Цей тип зв'язку повинен бути асинхронним. Прикладом є механізм публікації / підписки, який використовується в шаблонах типу Event-driven архітектури. Він заснований на інтерфейсі шини подій або на брокера повідомлень при поширенні оновлень даних між декількома мікросервісамі через події; зазвичай він реалізується через сервісну шину або подібний артефакт, такий як сервісна шина Azure, використовуючи теми і підписки.

Спільнота мікрослужб просуває філософію "розумних кінцевих точок і німих труб" Це гасло заохочує конструкцію, яка максимально розв'язана між мікрослужбамі, і настільки связна, наскільки це можливо, всередині однієї мікрослужби. Як пояснювалося раніше, кожна мікрослужба володіє своїми власними даними і власною логікою домену. Але мікроуслугі, складові наскрізне додаток, зазвичай просто хореографіруются за допомогою комунікацій REST, а не складних протоколів, таких як WS- * і гнучких комунікацій, керованих подіями, замість централізованих бізнес-процесів-орхестраторов.

Двома зазвичай використовуваними протоколами є HTTP запит / відповідь з ресурсними API (коли запит робиться найчастіше), і легка асинхронна передача повідомлень при обміні оновленнями між декількома мікрослужбамі. Більш докладно вони пояснюються в наступних розділах.

7 АРХІТЕКТУРА СИСТЕМИ

Виходячи з огляду засобів програмної реалізації та методів розробки була створена модель архітектури для підсистеми індивідуальних сторінок викладачів, що реалізує функціонал поставлений в задачах роботи та відповідає висунутим вимогам. Загальна система складається з головних сервісів та допоміжних. До головних сервісів відносяться:

- Сервіс викладачів;
- Сервіс студентів.

До допоміжних:

- Сервіс розкладу;
- Сервіс аутентифікації;
- Сервіс розподілу навантаження(шлюз);
- Сервіс брокера повідомлень;
- Сервер управління базою даних;
- Сервер управління об’єктним сховищем.

Структурна схема системи показана на рисунку 7.1.

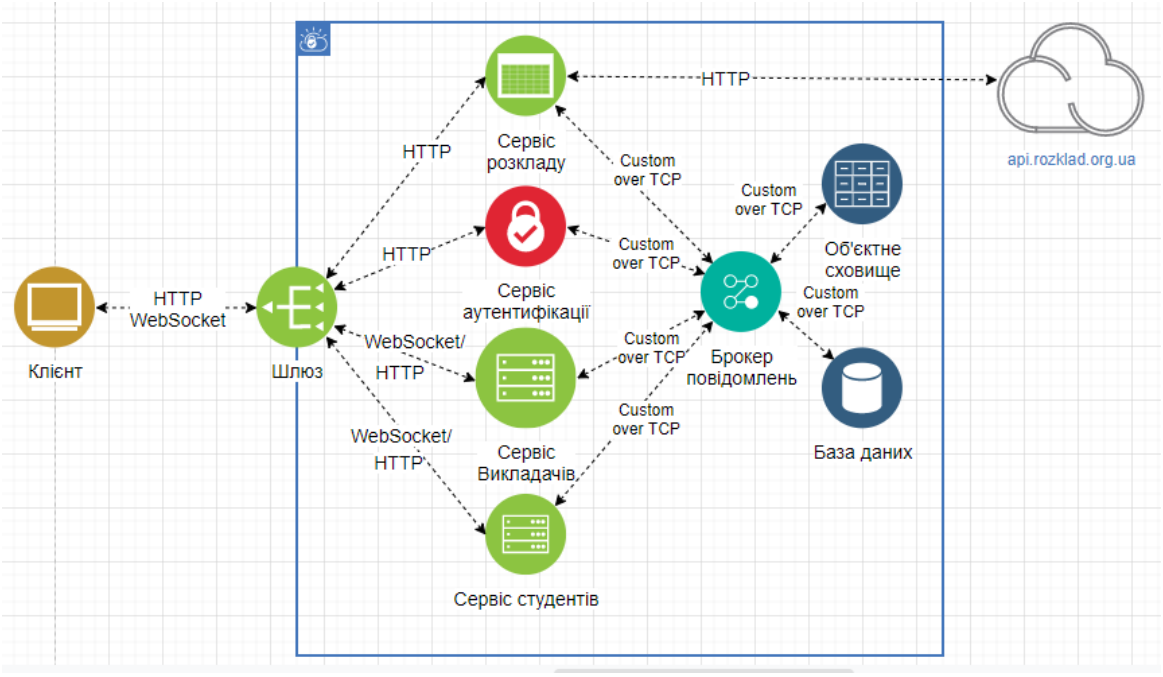

Рисунок 7.1 - Структурна схема системи

7.1 Сервіс студентів

Даний сервіс є одним із головних і являє собою підсистему, яка є об’єктом даної роботи. Він спілкується з іншими сервісами та обробляє запити, які пов’язані зі персональною сторінкою викладача, такі як:

- Надання інформації про студента, його розклад, доступні електронні матеріали, предмети, які він вивчає, та їх викладачі;
- Комунікація з об’єктним сховищем для надання учбових матеріалів;
- Синхронізація даних з сервісом розкладу.

Підсистема тісно пов’язана як з підсистемою викладачів, так і з сервісом розкладу та брокером повідомлень.

7.2 Сервіс розкладу

Головна задача сервісу розкладу полягає у обробці та наданні інформації про розклад викладачів іншим сервісам системи.

Під час запуску системи сервіс надсилає запит до api.rozklad.org.ua для отримання списку усіх викладачів, інформація про яких доступна на сайті. Отримавши список усіх викладачів, сервіс надсилає запити для отримання розкладу кожного викладача. Оскільки структура даних, що надходять, містить аномалії та непотрібну сервісу інформацію, ці дані проходять обробку з метою виділення та реструктурізації необхідної інформації. Отримані після обробки дані, зберігаються у кеші системи, що дозволяє не робити додаткові запити, з подальшою їх обробкою, за необхідності надати інформацію іншому сервісу підсистеми. Кеш сервісу оновлюється раз на добу.

Після автентифікації користувач отримує JWT токен, який клієнт використовує для отримання доступу до сервісу розкладу. Цей токен містить інформацію про ім’я та роль користувача у системі. В залежності від ролі користувача сервіс надає йому різні можливості. У системі доступні 4 ролі: «Адміністратор», «Староста», «Студент» та «Викладач». Користувачу авторизованому як «Студент» доступні такі функції:

- Перегляд розкладу групи;
- Перегляд опису предмету;

- Перегляд розкладу викладача;

Діаграма користування сервісу розкладу відображена на рисунку 7.2.

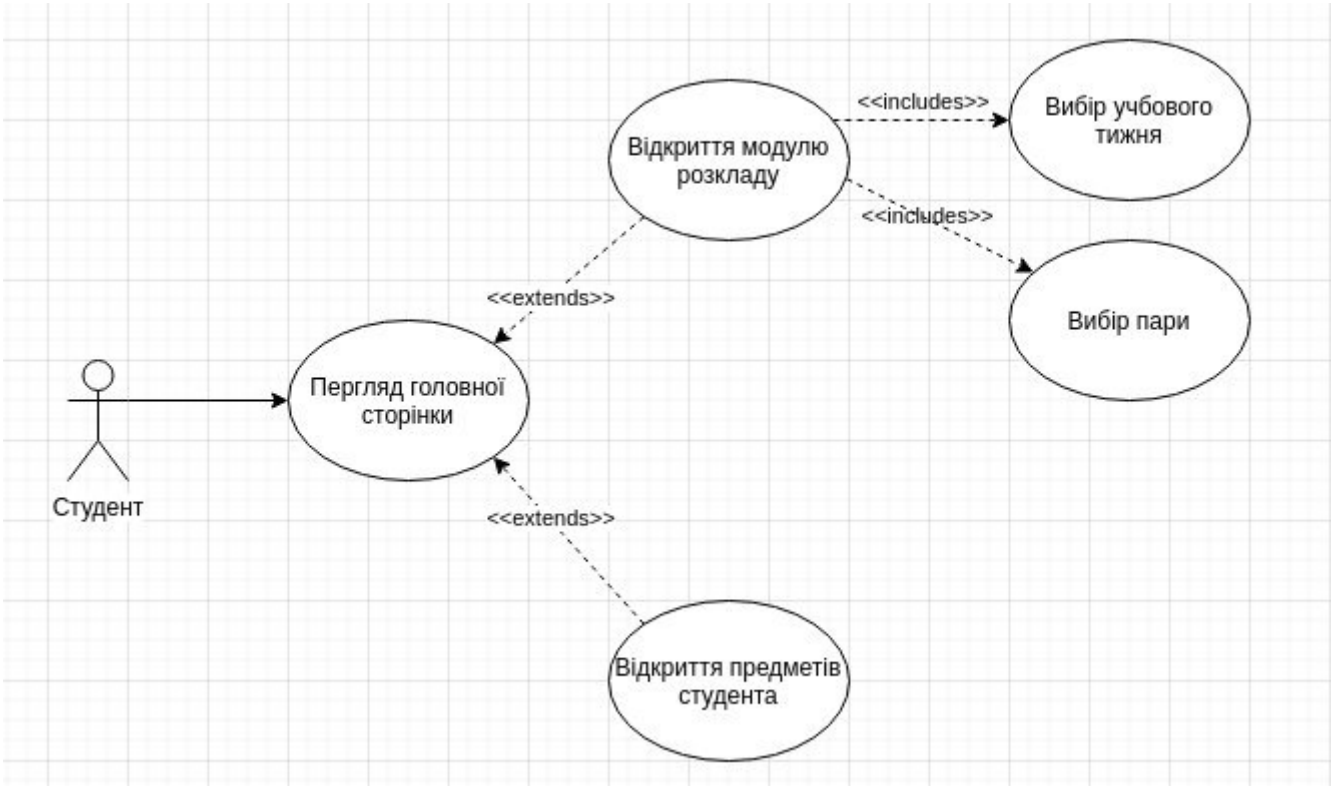

Рисунок 7.2 – Use Case діаграма студента комунікації студента з сервісом розкладу

7.3 Сервер брокеру повідомлень

Брокер повідомлень використовуються для пересилки повідомлень між окремими сервісами. Брокер дозволяє розсилати повідомлення для груп сервісів та забезпечує гарантію доставки повідомлення до кожного сервісу групи. Так як сервіси можуть бути перевантажені або недоступні з інших причин, брокери повідомлень мають механізми збереження та пересилки повідомлень, алгоритми роботи яких дозволяють упевнитися у доставці повідомлення та при цьому не впливають на роботу сервісів.

У якості сервера брокера повідомлень системи було обрано платформу Apache Kafka. Платформа надає API для взаємодії з нею сервісів у якості двох ролей: видавець та підписник. Видавці надсилають повідомлення у теми, підписники яких мають отримати це повідомлення. Після підписки сервісу на тему для нього створюється персональна черга для повідомлень, отже сервіс гарантовано отримає

повідомлення, коли буде готовий. Схема роботи Apache Kafka зображено на рисунку 7.3.

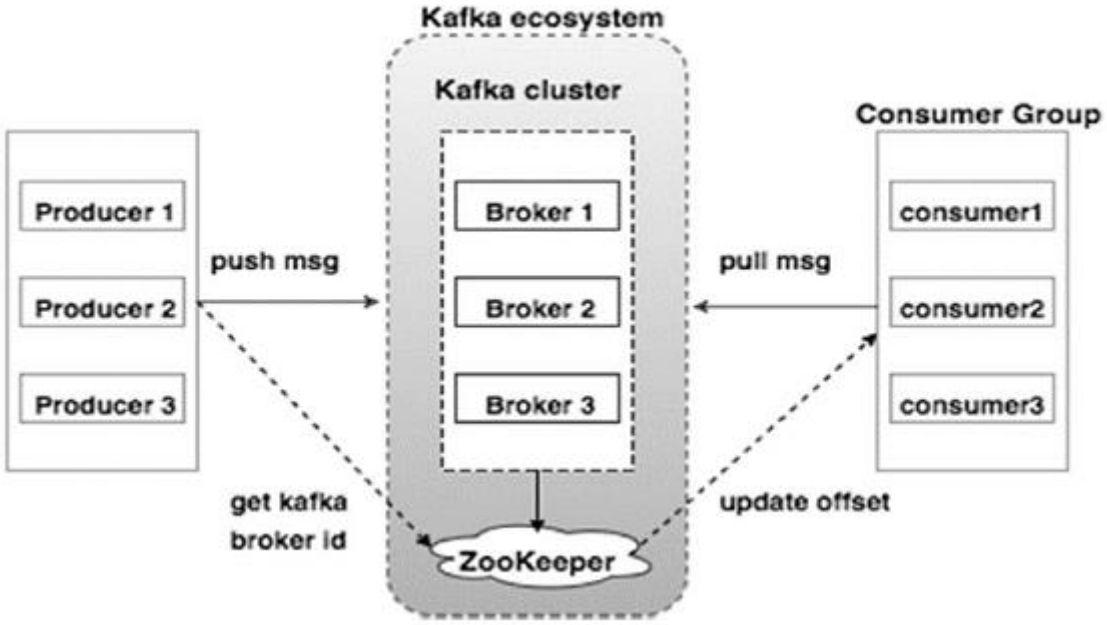

Рисунок 7.3 - Схематичне зображення роботи Apache Kafka

8 ПРОГРАМНА РЕАЛІЗАЦІЯ ПІДСИСТЕМИ

Програмна реалізація підсистеми складається з двох компонентів. Бібліотеки класів моделей бізнес-логіки та самого коду підсистеми. Таку структуру було впроваджено через те, що комунікація між мікро-сервісами здійснюється за допомогою сеарелізації моделей об’єктів в JSON формат. Тому для уникнення дуплікації коду проміжних моделей бізнес логіки, було вирішено винести ці моделі в окрему бібліотеку, якою будуть користуватися всі підсистеми.

8.1 Структура проекту

Структура проекту (рис. 8.1) в загалом поділяється на шість основних пакунків:

- Контролери – відповідають за первинну обробку HTTP запитів REST API;
- Репозиторії – відповідають за роботу з реляційною базою даних;
- Сервіси – є проміжним шаром між контролерами та репозиторіями. Реалізують управління об’єктами бізнес-логіки;
- Веб-сокети – пакунок містить класи, що відповідають за повну обробку та управління інформацією передану через протокол WebSocket.
- Повідомлення – пакунок містить клієнти та споживачі повідомлень брокера повідомлень Apache Kafka;
- Утіліти – класи шо мають специфічний функціонал, та використовуються у різних шарах чи рівня підсистеми.

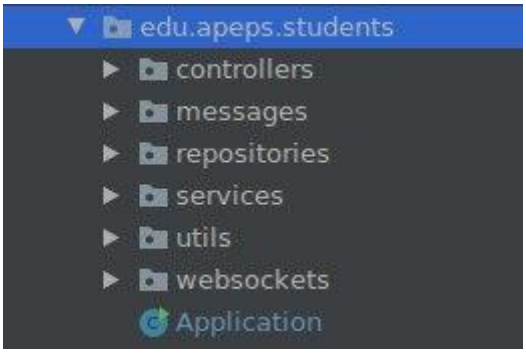

Рисунок 8.1 – Структура проекту

8.2 Бізнес-модель

Бібліотека бізнес-моделей edu.apeps.models містить в собі чотири пакунки на кожну підсистему та сервіс: auth, schedule, teachers та students. Пакунок students

описує бізнес-моделі даної підсистеми.

Зв’язки між класами об’єктів зображені на рисунку 8.2 та додатку (). Також даний пакунок має зв’язок з пакунком auth, так-як в ньому описані об’єкти авторизації користувачів системи, які пов’язані з моделлю студента.

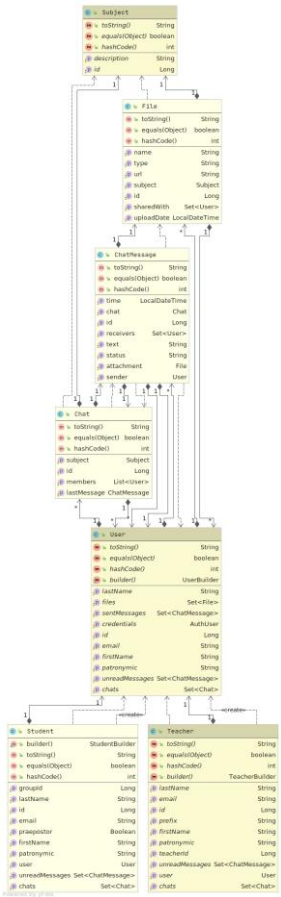

Рисунок 8.2 – UML діаграма класів моделі бізнес логіки

Головні класи пакунку:

- User – містить загальну інформацію про користувача;
- Student – містить інформацію про групу, та чи поле позначення статусу старости. Також має зв’язок один-до-одного з класом User;
- Subject – містить опис об’єкту предмета;
- Chat – містить опис об’єкту чата. Має зв’язки з один-до-одного з Subject, один-до-багатьох з ChatMessage та багато-до-багатьох з User;
- File – містить опис об’єкту файлу;
- ChatMessage – опис об’єкту повідомлення в чаті.

8.3 Структура бази даних

База даних містить в собі 11 таблиць, 8 з яких містять безпосередньо дані об’єктів, та 3 для реалізації зв’язків багато-до-багатьох між ними. Схему бази даних зображено на рисунку 8.3

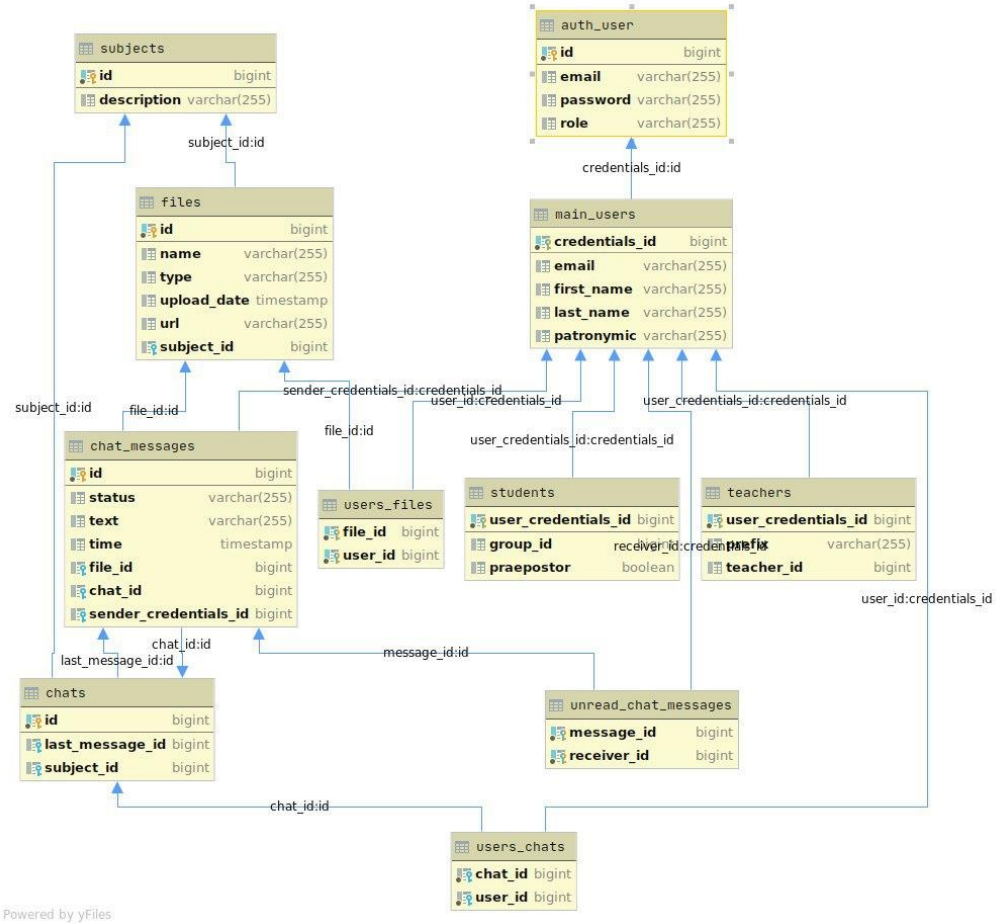

Рисунок 8.3 – Схема public бази даних підсистеми

9 МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПІДСИСТЕМОЮ

Розроблена система представляє собою веб-сайт який надає графічний інтерфейс для користувачів, та доступ до REST API, через саб-домен api. Це дозволяє іншим розробникам реалізовувати свої графічні клієнти для системи та спрощує доступ до сервісу з інших програмних додатків.

Для користувачів, які не автентифіковані в системі можливий доступ лише до сторінки логіну та сторінки встановлення паролю. При будь-який спробах перейти на сторінки, які потребують автентифікації, користувач буде переправлений на сторінку логіну.

Після успішною автентифікації викладач потрапляє на головну сторінку сервісу (рис. 9.1).

Рисунок 9.1 – Головна сторінка студента

На головній сторінці студента, користувачу доступні два меню. Меню з права містить переходи до розкладу занять, список груп, у яких викладач проводить заняття. У меню зліва містить кнопки переходу до модулів Матеріали, Чати та Предмети. На центральній частині модулю Головна розташована інформація про викладача.

При переході на модуль Матеріли, в центральній частині екрану з’являється список матеріалів, які доступні студенту, згруповані по предметам(рис. 9.2). При

натискані на файл, починається процес завантаження його на персональний комп'ютер користувача. При натисканні на іконку контекстного меню, зв'являється можливість переслати файл у чат. Для завантаження файлу достатньо перетягнути файл до списку матеріалів, після чого з'явиться модульне вікно, в якому потрібно буде вказати, до якого предмету цей файл належить.

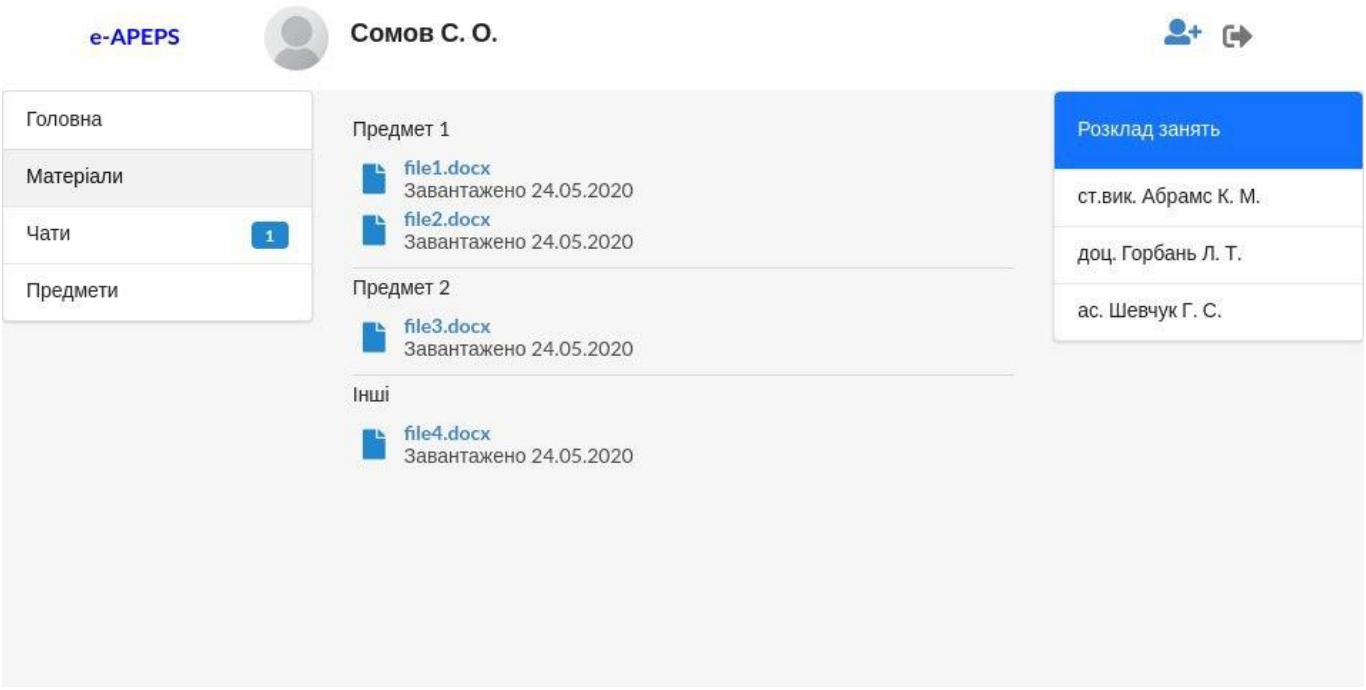

Рисунок 9.2 – Вигляд модулю Матеріали

При переході на модуль Чати (рис. 9.3), в центральній частині екрану з'являється список чатів. Кожна з карток чату містить його назву або назву предмету, до якого відноситься даний час, час останнього повідомлення, відправника цього повідомлення та частину тексту повідомлення. Також на кожній з карток міститься кількість не прочитаних повідомлень користувача.

Модуль Предмети (рис. 9.4)містить в собі список предметів студента, які він викладає. Кожний елемент списку містить назву предмету, перелік груп та короткий опис предмету. При натисканні на кожен з предметів, користувач переходить до сторінки повного опису предмету.

Також, кожна з підсистем містить шапку, в якій відображається фото користувача, та кнопки додому та виходу. При натисканні на мініатюру фото користувача, користувач переходить до сторінки редактування профілю.

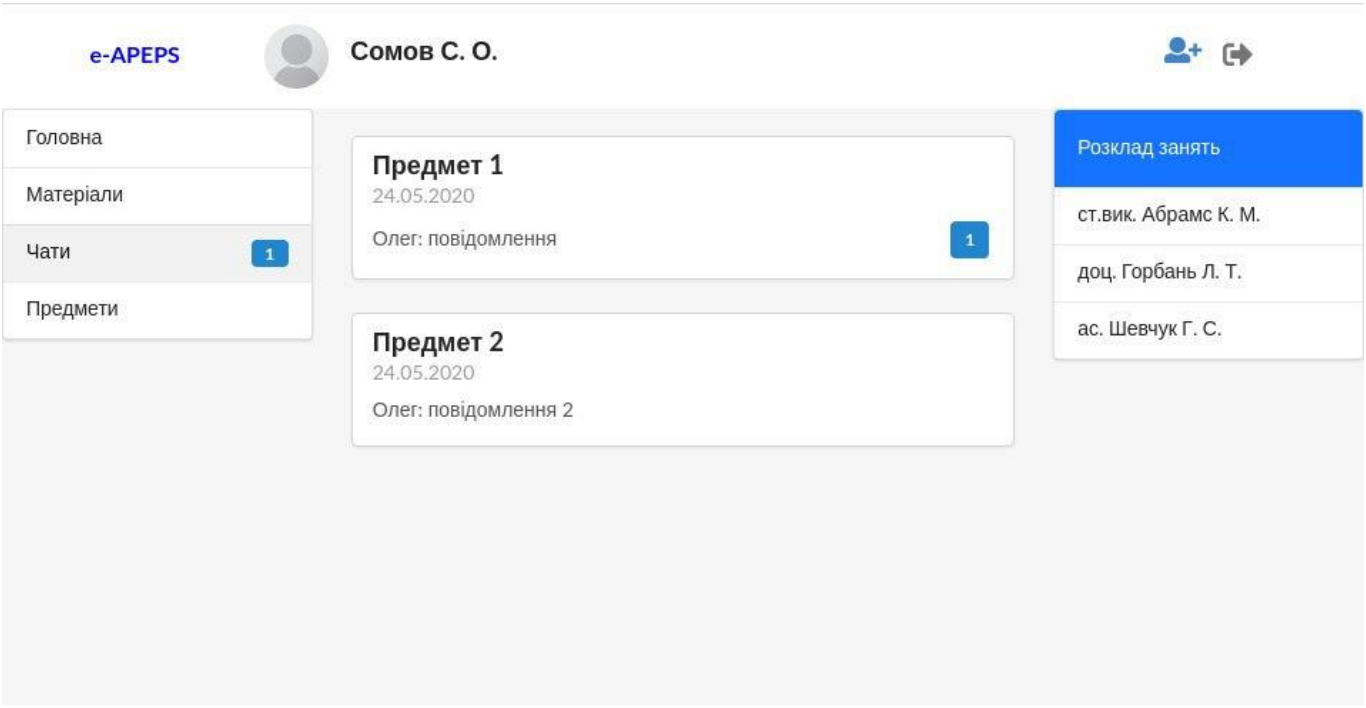

Рисунок 9.3 – Вигляд модулю Чати

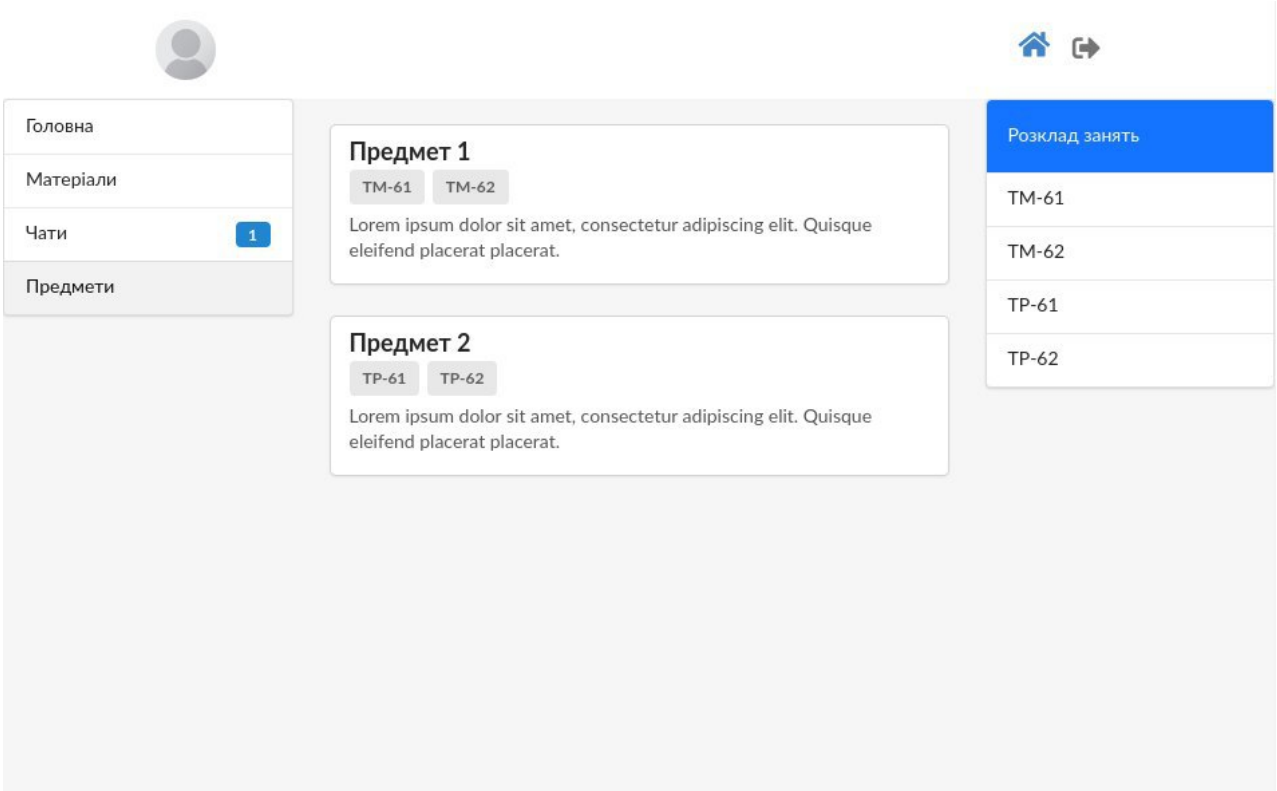

Рисунок 9.4 – Вигляд модулю Предмети

ВИСНОВКИ

Дана дипломна робота описує проектування та створення підсистеми індивідуальних сторінок студентів в загальній системі електронного кампусу для комунікації між студентами і викладачами та підтримки навчального процесу. В умовах сьогодення, при вимушеному дистанційному навчанні, гостро постає проблема комунікації та взаємодії між викладачами та студентами. На разі, для вирішення цієї проблеми університет використовує стару систему підтримки навчального процесу, яка не відповідає вимогам які ставить дана ситуація. Також, в останні роки росте тенденція впровадження корпоративних систем у веб інфраструктуру завдяки створенню сервісів, які надають інструменти для легкого розгортання програмних додатків у хмарних провайдерів. Тому реалізація нової системи, яка інтегрується з вже існуючою та вирішує її недоліки, є як ніколи актуальною.

Як результат виконання роботи є детальний опис створення підсистеми індивідуальних сторінок студентів та її реалізація. Всі завдання та більшість вимог до системи були виконані під час розробки.

Були створені головні моделі бізнес логіки, що описують область роботи підсистеми, зв’язки між ними та сервіси для їх обробки. Головними моделями є класи студента, предмету, групи, чату, повідомлень та файлових матеріалів.

Був створений сервіс інтеграції для інтеграції з існуючою системою. Для реалізації були описані відображення моделей підсистеми з моделями системи розкладу занять НТУУ «КПІ» ім. І. Сікорського, такі як заняття, викладач та предмет. Також була розроблений сервіс синхронізації даних даної підсистеми з існуючою системою університету.

Був створений функціонал, який задовольняє потреби до підтримки навчального процесу зі сторони студента: відображення розкладу занять, можливість завантажувати матеріал на персональний комп’ютер та можливість комунікації з викладачами.

Був розроблений зручний графічний інтерфейс користувача, з акцентом на максимально інтуїтивне розташування компонентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронна система Moodle [Електронний ресурс] – Режим доступу: https://moodle.org/.
2. Електронний кампус КПІ ім. І. Сікорського [Електронний ресурс] – Режим доступу: https://ecampus.kpi.ua.
3. The GNU Operating System [Електронний ресурс] – Режим доступу: https://www.gnu.org.ua/.
4. Java [Електронний ресурс] – Режим доступу: https://www.java.com/.
5. Spring Framework [Електронний ресурс] – Режим доступу: https://spring.io//.
6. Micronaut [Електронний ресурс] – Режим доступу: https://micronaut.io/.
7. Spock Framework [Електронний ресурс] – Режим доступу: http://spockframework.org/.
8. JavaScript language [Електронний ресурс] – Режим доступу: https://www.javascript.com/.
9. React.js [Електронний ресурс] – Режим доступу: https://reactjs.org/.
10. Брокер повідомлень Apache Kafka [Електронний ресурс] – Режим доступу: https://kafka.apache.org/.
11. Grails [Електронний ресурс] – Режим доступу: https://grails.org/.
12. Server Netty [Електронний ресурс] – Режим доступу: https://netty.io/.
13. MIME Specification RFC 1521 [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc1521/.
14. MIME Specification Part 2 RFC 1522 [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc1522/.
15. The WebSocket Protocol [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc6455.
16. MinIO [Електронний ресурс] – Режим доступу: https://min.io/.
17. APACHE LICENSE, VERSION 2.0 [Електронний ресурс] – Режим доступу: https://www.apache.org/licenses/LICENSE-2.0.
18. Електронна система e-APEPS [Електронний ресурс] – Режим доступу: https://e-

apeps.ml/.
19. Erik Wilde, Cesare Pautasso. REST: From Research to Practice. — Springer Science & Business Media, 2011. — 528 p
20. The WebSocket Protocol [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc6455.

ДОДАТОК А

Розробка системи взаємодії індивідуальних сторінок викладачів та студентів.
Підсистема: індивідуальні сторінки студентів

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20Б

Аркушів 1

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б ПЗ	Записка.docx	Текстова частина дипломної роботи
Комплекс		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б	teachers	Підсистема індивідуальних сторінок викладачів.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б	students	Підсистема індивідуальних сторінок студентів. Розроблена у даній роботі
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б К1	client	Основна папка, яка містить файли для клієнта.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б К2	students	Основна папка, яка містить файли для сервісу студента.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б К4	teachers	Основна папка, яка містить файли для сервісу викладача.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б К5	schedule	Основна папка, яка містить файли для сервісу розкладу.
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20 Б К6	auth	Основна папка, яка містить файли для сервісу розкладу.

ДОДАТОК Б

Розробка системи взаємодії індивідуальних сторінок викладачів та студентів.
Підсистема: індивідуальні сторінки студентів

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20Б

Аркушів 4

Київ 2020

```
package edu.apeps.main.controllers;

import edu.apeps.main.services.UserService;
import edu.apeps.models.general.Student;
import edu.apeps.models.general.Teacher;
import edu.apeps.main.repositories.*;
import edu.apeps.models.general.User;
import io.micronaut.http.MediaType;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import io.micronaut.http.annotation.Put;
import io.micronaut.security.annotation.Secured;
import io.reactivex.Maybe;

import javax.inject.Inject;
import javax.inject.Singleton;
import java.security.Principal;

@Controller("/user")
@Secured("isAuthenticated()")
public class UserController {
    @Inject
    UserService userService;


    @Get("/")
    public Maybe<User> index() {
        return userService.getUser();
    }

    @Get("/student")
    @Secured({"STUDENT","PRAEPOSTOR"})
    public Maybe<Student> student() {
        return userService.getStudent();
    }

    @Get("/teacher")
    @Secured({"TEACHER"})
    public Maybe<Teacher> teacher() {
        return userService.getTeacher();
    }

    @Put(uri = "/student", consumes = {MediaType.APPLICATION_JSON})
    @Secured({"STUDENT","PRAEPOSTOR"})
    public Maybe<Student> editStudent(Student student) {
        return userService.editStudent(student);
    }

    @Put(uri = "/teacher", consumes = {MediaType.APPLICATION_JSON})
    @Secured({"TEACHER"})
    public Maybe<Teacher> editTeacher(Teacher teacher) {
        return userService.editTeacher(teacher);
```

```
    }
}
```

```
package edu.apeps.main.controllers;


import edu.apeps.main.services.ChatService;
import edu.apeps.main.services.SubjectService;
import edu.apeps.models.general.Chat;
import edu.apeps.models.general.Subject;
import io.micronaut.http.annotation.*;
import io.micronaut.security.annotation.Secured;
import io.reactivex.Flowable;
import io.reactivex.Maybe;
import org.reactivestreams.Publisher;

import javax.annotation.Nullable;
import javax.inject.Inject;
import java.util.List;

@Controller("/chats")
@Secured("isAuthenticated()")
public class ChatController {
    @Inject
    ChatService chatService;

    @Get("/{?ids}")
    @Secured({"STUDENT","PRAEPOSTOR","TEACHER"})
    public Publisher<Chat> get(@Nullable @QueryValue("ids") List<Long>
ids) {
        assert ids != null;
        return Flowable.fromIterable(ids).flatMapMaybe(id ->
chatService.get(id));
    }

    @Put("/")
    @Secured({"TEACHER"})
    public Maybe<Chat> update(Chat chat) {
        return chatService.update(chat);
    }

    @Post("/")
    @Secured({"TEACHER"})
    public Maybe<Chat> create(Chat chat) {
        return  chatService.update(chat);
    }
}
```

```
package edu.apeps.main.websockets;

import edu.apeps.main.caches.ChannelCache;
import edu.apeps.main.services.MessageService;
```

```
import edu.apeps.models.general.ChatMessage;
import io.micronaut.websocket.WebSocketSession;
import io.reactivex.Maybe;

import javax.inject.Inject;
import javax.inject.Singleton;
import java.security.Principal;
import java.util.HashSet;
import java.util.stream.Collectors;

@Singleton
public class WebSocketService {


    @Inject
    private ChannelCache channelCache;

    public Maybe<Channel> getChannel(Long id, ChatMessage message,
WebSocketSession session){
        return channelCache.getFromCache(message.getChat())
                .defaultIfEmpty(
                        Channel.builder()
                                .id(message.getChat())
                                .sessionId( new HashSet<>(){
                                                {

add(session.getId());
                                                }
                                            }
                                )
                                .users( new HashSet<>(){
                                                {

add(message.getSender().getId());
                                                }
                                            }
                                )
                                .build()
                )
                .filter(any -> id.equals(message.getSender().getId()))
                .flatMap( channel -> {
                    var openSessionIds =
session.getOpenSessions().stream()
                            .map(WebSocketSession::getId)
                            .collect(Collectors.toSet());
                    channel.getSessionId().add(session.getId());

                    channel.setSessionId(
                            channel.getSessionId().stream()
                            .filter(openSessionIds::contains)
                            .collect(Collectors.toSet())
                    );
```

```
channel.getUsers().add(message.getSender().getId());
                    return
channelCache.putIntoCache(channel.getId(),channel);
                });
    }
}
```

```
package edu.apeps.models.general;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import io.micronaut.core.annotation.Introspected;
import lombok.*;
import lombok.experimental.Delegate;

import javax.persistence.*;

@Getter @Setter @RequiredArgsConstructor
@ToString(doNotUseGetters = true)
@EqualsAndHashCode(doNotUseGetters = true)
@AllArgsConstructor
@Builder
@JsonIgnoreProperties(value = { "user",
"files","sentMessages","credentials"})
public class Teacher{
    private Long id;
    private Long teacherId;
    private String prefix;

    @Delegate(excludes = User.UserExclude.class)
    @Builder.Default
    private User user = new User();
}
```

ДОДАТОК В

Розробка системи взаємодії індивідуальних сторінок викладачів та студентів.
Підсистема: індивідуальні сторінки студентів

Опис програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ6283_20Б

Аркушів 7

Київ 2020

АННОТАЦІЯ

Даний додаток містить опис мікросервіс системи взаємодії індивідуальних сторінок викладачів та студентів, підсистеми індивідуальних сторінкок студентів. Розроблено на базі фреймворку Micronaut для надання можливості комунікації викладачів зі студентами та підтримки навчального процесу. Створено веб-сервіс мікросервісного типу. Сервіс виконує наступні завдання:

- Надання розкладу;
- Редагування опису предметів;
- Можливість спілкування у чаті групи;
- Надсилання результату обрахунків на клієнт.

При розробці веб-сервісу використовувався мова програмування Java, фреймворк Micronaut та бібліотека React.js (на мові програмування Javascript).

ЗМІСТ

1. Загальні відомості 52
2. Функціональне призначення 53
3. Опис логічної структури 54
4. Технічні засоби, що використовуються 55
5. Виклик і завантаження 56
6. Вхідні і вихідні дані 57

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис веб-сервісу з підсистемою індивідуальних сторінок студентів системи e-APEPS. У додатку Б міститься програмний код головних модулів розробленого серверу. Веб-сервіс виконано для подальшої інтеграції на серверну віддалену машину. Для коректної роботи серверу необхідно встановити всі залежні модулі на сервер.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений мікросервіс виконує завдання надання розкладу, редагування опису предметів, можливість спілкування у чаті групи, надсилання результату обрахунків на клієнт. Розроблений сервіс може використовуватись в якості системи підтримки навчання, надання учбових матеріалів та комунікації між студентами та викладачами. Функціональних обмеження на використання веб-сервісу полягає лише в його апаратному забезпеченні.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Вся система поділяється на 8 компонентів: сервіс викладачів, сервіс студентів, сервіс розкладу, сервіс аутентифікації, сервер брокера повідомлень, сервер об’єктного сховища, сервер управління базою даних та розподільник навантаження. Підсистема що розроблялася у даному дипломі відповідає за обробку інформації пов’язаною з функціоналом сторінки студентів.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для запуску та тестуванні системи у наближених до реальних умовах було використано сервіси від Google Cloud Platform. Завдяки можливостям фреймворку Micronaut, було розгорнуто і впроваджено систему повністю з мінімальними налаштуваннями.

Розроблений веб-сервіс працює на будь-якій платформі у будь-якому браузері. У подальшому необхідно підняти веб-сервіс на віддаленому сервері для його стабільної роботи та доступу у мережі інтернет.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для запуску всіх сервісів потрібно інсталювати наступне програмне забезпечення: docker, nginx та npm. Для запуск серверу клієнта та сервісів автентифікації, розкладу, викладачів та студентів необхідно виконати команду "./docker-build.sh" у кожному з каталогів з вихідним кодом сервісів.

Також з головної директорії проекту необхідно перенести конфігураційний файл для розподільника навантаження nginx.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними підсистеми є інформації про викладачів, матеріали та файли, повідомлення та інша інформація. Так-як система займається обробкою та збереженням даних, вихідні дані в більшості збігаються з вхідними і невелику кількість додаткової інформації.